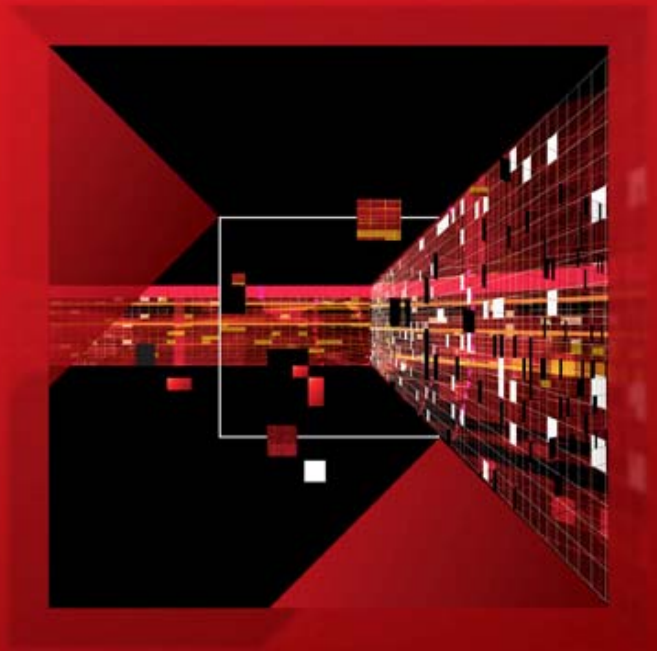
ADVANCED GRAPHICS FUNCTIONALITY ON WINDOWS USING DIRECTX

Michael Oneppo moneppo@microsoft.com)
Program Manager II
Microsoft Corporation

AGENDA

- Direct2D Recap
- Direct2D Performance
- Interoperability
- Development Recommendations
- DirectWrite
- What's new in Windows 7 SP1

WHAT IS DIRECT2D?



DIRECT2D

- GPU-Accelerated 2D Rendering Library
- Better quality and performance
- Supports server side rendering – good SW performance
- Provides interoperability with GDI/GDI+ code bases
- Works well with sister DirectX APIs like D3D

DIRECT2D API

- Win32, interface-based API
  - Consistent with Direct3D
  - Primarily used via C++
- Immediate mode
  - Create resources up front
  - Re-use resources frame over frame
  - Manage resources per graphics adapter

SAMPLE USAGE

- Create top-level objects
  - `D2D1CreateFactory()` → `ID2D1Factory`
  - Factory → `ID2D1HwndRenderTarget`
  - Also bitmap, interop, and intermediate RTs
- Create resources
  - Factory → `ID2D1Geometry`
  - Also bitmaps, layers
- Draw primitives using render target
  - `BeginDraw()`
  - `FillGeometry()`
  - `EndDraw()`

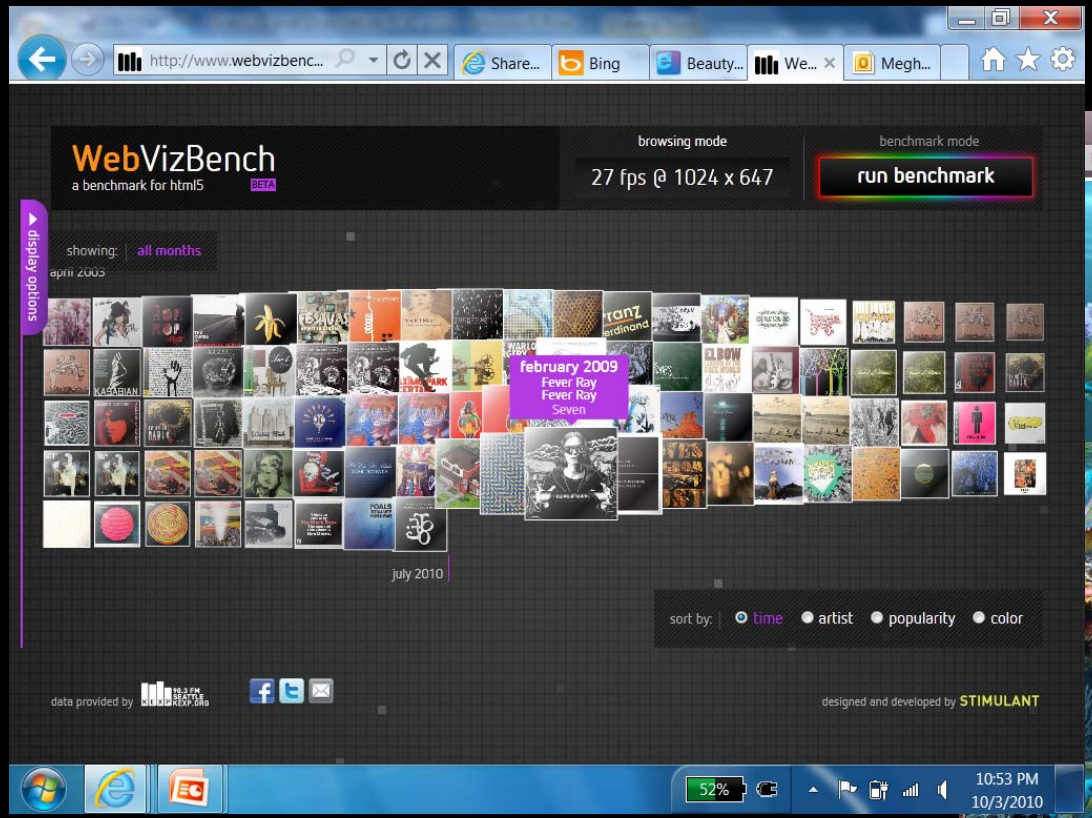
DIRECT2D API

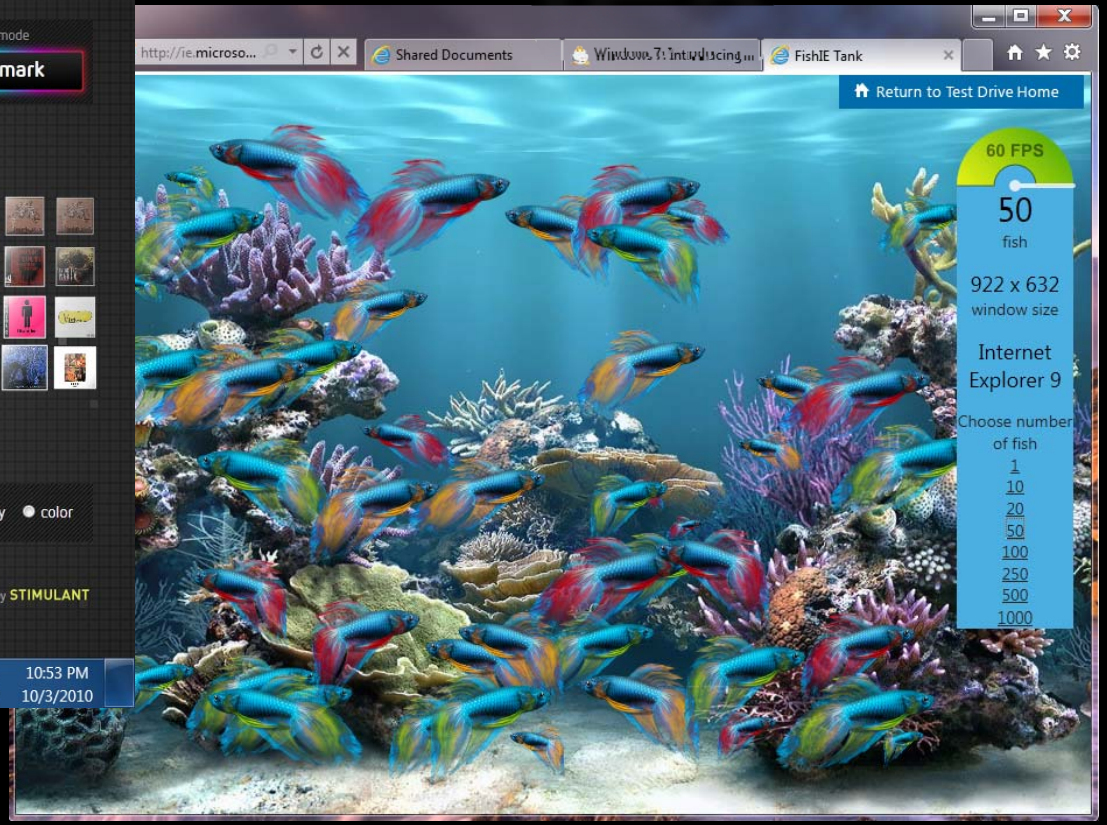
DRAWING RESOURCES

- Resolution-independent coordinate system
  - Floating point values
  - Affine transforms
- Device-independent resources
  - Geometries
  - Created via `ID2D1Factory`
- Device-dependent resources
  - Brushes, bitmaps, intermediate render targets
  - Created via `ID2D1RenderTarget`
  - Become invalid if the target becomes invalid

WHAT'S POSSIBLE

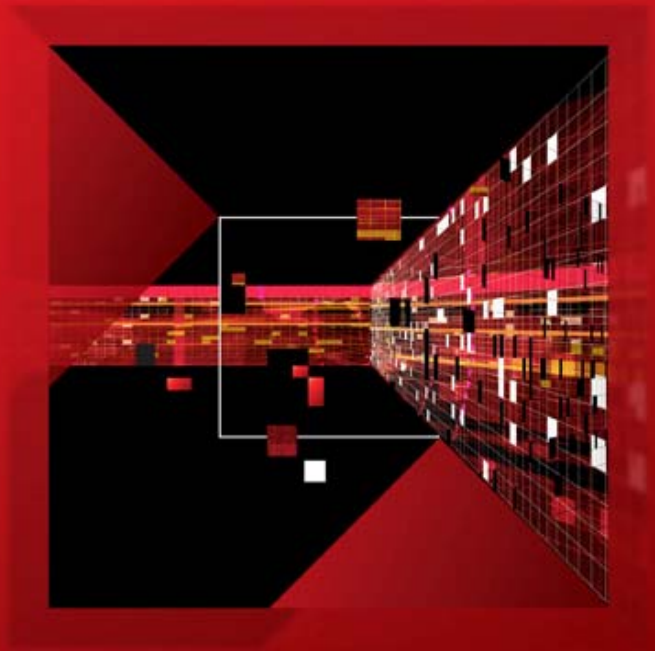
IE9





WHAT'S POSSIBLE

2D PERFORMANCE



PERFORMANCE AND DIRECTX

- Direct2D built atop Direct3D 10
- Allocation overhead in Direct3D 10 is high

Reduce:
Controlling
scene complexity

Recycle:
Repurposing already
created resources

Reuse:
Caching intermediate data
to reduce GPU work

- A8 resources and grayscale caching
- Mesh & mesh caching

Reuse:
Caching intermediate data to reduce GPU work

A8 RESOURCES

- Great way to cache text, opacity maps, or otherwise grayscale content
- Use CreateCompatibleRenderTarget with DXGI_PIXEL_FORMAT_A8_UNORM
- Use FillOpacityMask to draw any A8 surface to a render target

CACHING MESHES

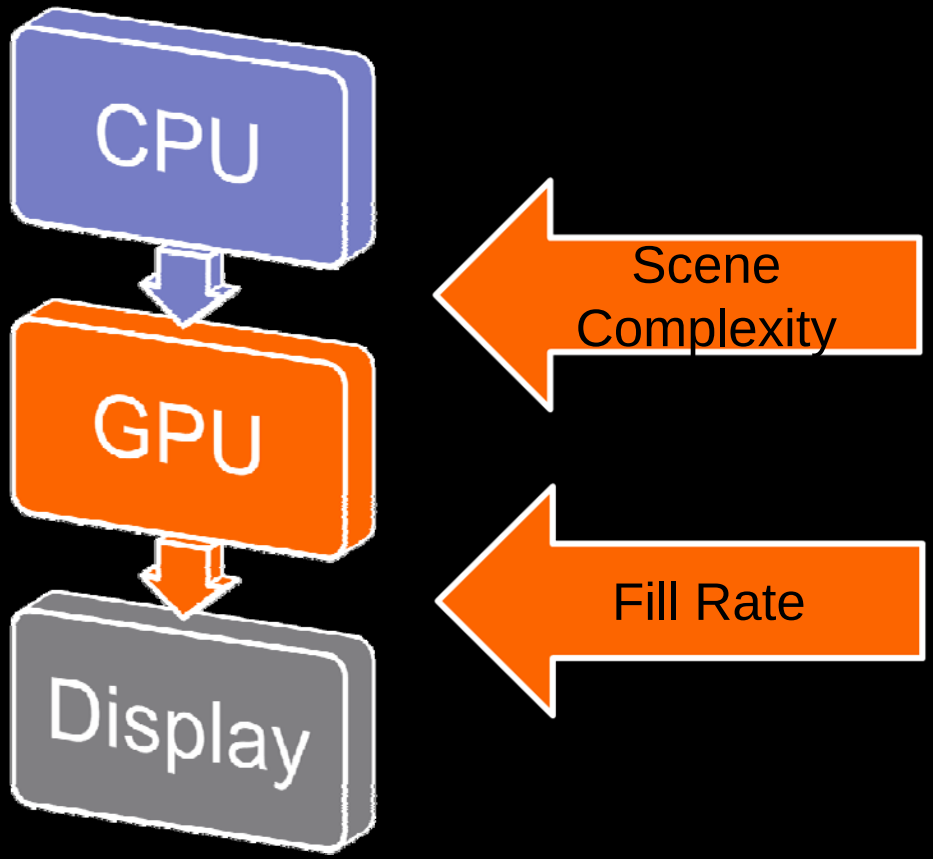
- ID2D1Mesh represents (aliased) triangles
  - Direct2D can render these very quickly
- ID2D1Mesh::Open returns an ID2D1TessellationSink
- Pass this to the ID2DGeometry::Tessellate function to fill the mesh.
- Use FillMesh to draw it
- Manually implement the ID2D1TessellationSink if you want to see the triangles
  - Best way to transfer geometry to Direct3D.

PERFORMANCE AND DIRECTX

- Fill-rate bound vs. scene complexity-bound
- Improving fill-rate bound cases
- Improving complexity-bound cases

Reduce:
Controlling scene complexity

MANAGING SCENE COMPLEXITY



- How do you know?
- Smaller render target improves performance?
  - Fill rate bound

IMPROVING FILL-RATE BOUND CASES

- Reduce render target size
  - –Find the exact number of pixels you need to render to
- Pay attention to over-draw
  - –Rendering elements that aren't visible
  - –Occlusion
  - –Clipping
- Complex brushes & effects

QUALITY VS. PERFORMANCE

- Scene complexity is increased by:
  - –Anti-aliasing
  - –Re-rendering of elements
  - –Sophisticated geometries

Recycle:
Repurposing already created resources

- Surface size and performance
- Using tiling or an “atlas”

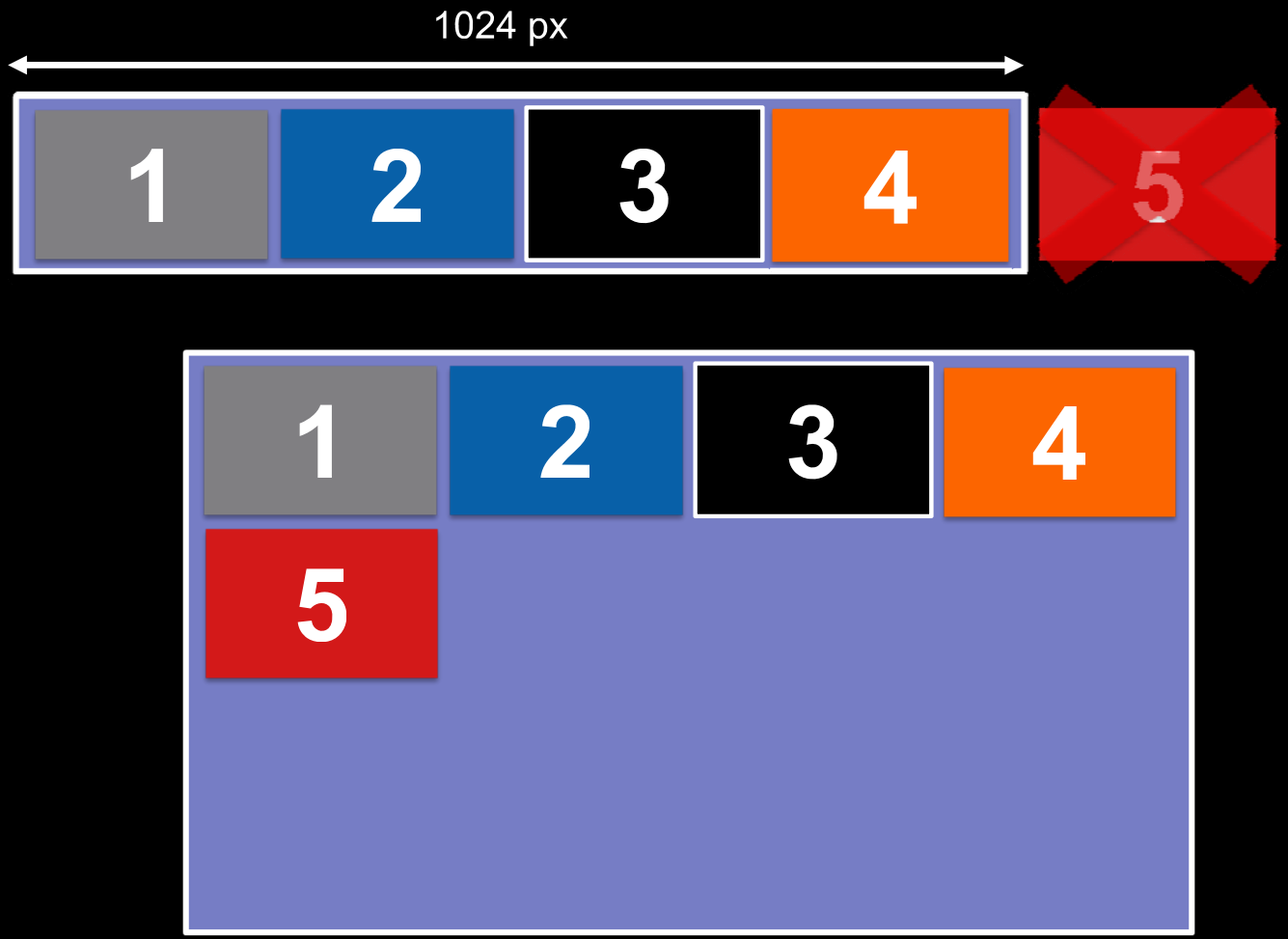
BIGGER SURFACES

- Seems counter-intuitive
  - Favor multi-purpose, larger textures over single use, smaller textures
  - Still only allocate what you need
- Minimum size 64Kb
  - For 24 bpp bitmaps, this is about 150x150
  - Recommended making 256x256 your global minimum

SURFACE SIZE

- Maximum size
  - Varies per card
  - Reasonable to assume 2048x2048, many modern cards support 4096x4096
- Find out using ID2D1RenderTarget::GetMaximumBitmapSize()
  - Size is in pixels

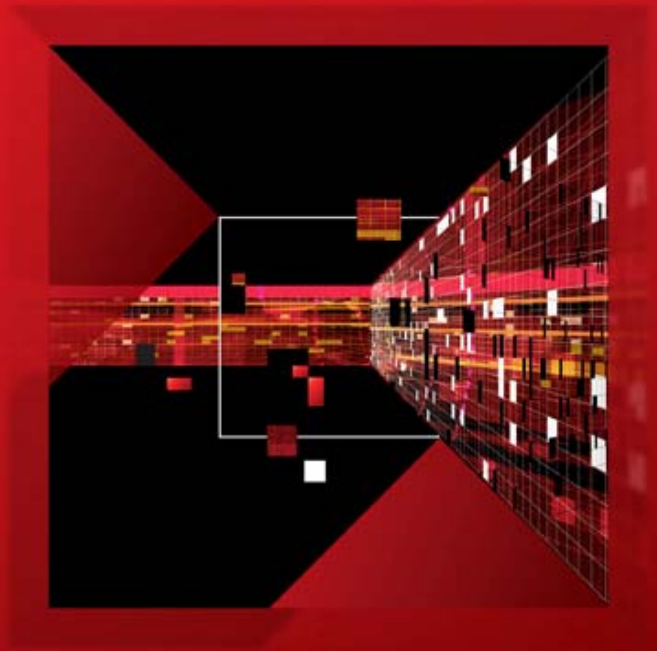
ATLASING



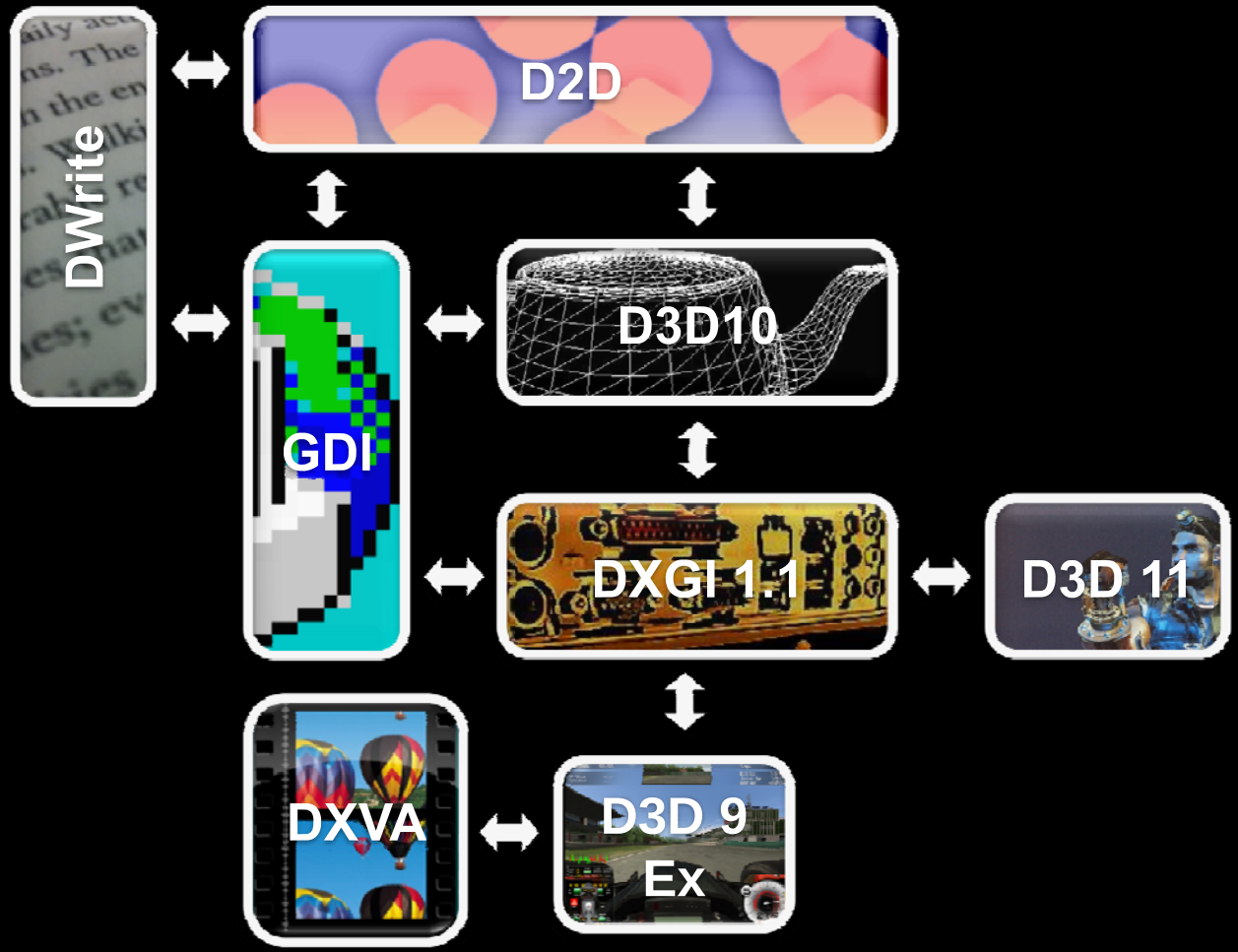
USING AN ATLAS

- You should create an “AtlasBitmap” class or struct
- Contains:
  - Reference to bitmap (pointer in C++)
  - Offset in X,Y
  - Size
- Provides you everything you need to use the DrawBitmap function
- Update atlas contents using the CopyFromBitmap function

DIRECT2D INTEROPERABILITY



RESOURCE SHARING

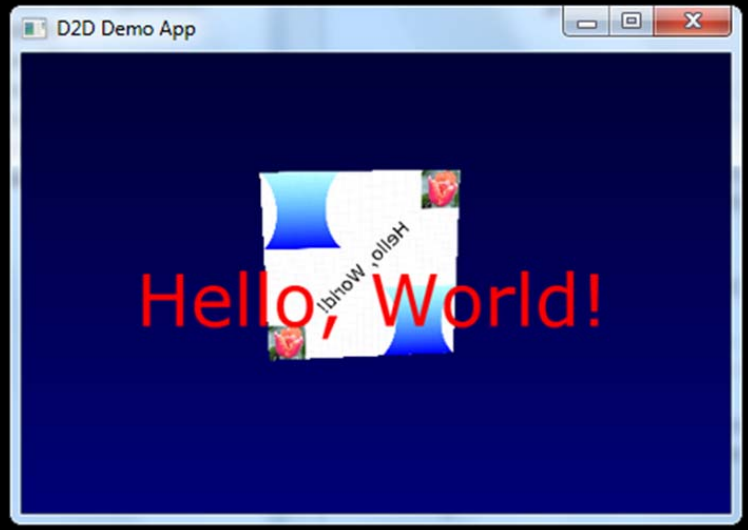


RENDER TARGET TYPES

- HWND Render Target
  - –“Basic” render target
  - –Can be software or hardware
- DXGI Render Target
  - –Draws to a application-supplied DXGI surface (i.e. swapchain)
- WIC Bitmap Render Target
  - –Draws to a WIC bitmap for saving the image to a file.

DXGI RENDER TARGETS

- **Use these whenever possible**
- Best way to get Direct2D and DirectWrite services into your 3D application



Direct2D and Direct3D10.1/11 device sharing

```
// Direct3D 10.1 Device and Swapchain creation
HRESULT D3D10CreateDeviceAndSwapChain1(
    __in IDXGIAdapter *pAdapter,
    __in D3D10_DRIVER_TYPE DriverType,
    __in HMODULE Software,
    __in UINT Flags,
    __in D3D10_FEATURE_LEVEL1 HardwareLevel,
    __in UINT SDKVersion,
    __in DXGI_SWAP_CHAIN_DESC *pSwapChainDesc,
    __out IDXGISwapChain **ppSwapChain,
    __out ID3D10Device1 **ppDevice
);
```

```
//get the back buffer of the swap chain
HRESULT
IDXGISwapChain::GetBuffer(
    [in] UINT Buffer,
    [in] REFIID riid,
    [in, out] void **ppSurface
);
```

Direct2D and Direct3D10.1/11 device sharing

```
//D2D Factory and RenderTarget creation
HRESULT WINAPI D2D1CreateFactory(
    __in D2D1_FACTORY_TYPE factoryType, __in REFIID riid,
    __in_opt const D2D1_FACTORY_OPTIONS *pFactoryOptions,
    __out void **ppIFactory );

HRESULT ID2D1Factory::CreateDxgiSurfaceRenderTarget(
    [in] IDXGISurface *dxgiSurface,
    [ref] const D2D1_RENDER_TARGET_PROPERTIES &renderTargetProperties,
    [out] ID2D1RenderTarget **renderTarget );

//Present the swap chain contents
HRESULT IDXGISwapChain::Present( [in] UINT SyncInterval,
    [in] UINT Flags );
```

GOING THE OTHER WAY

- Use CreateSharedBitmap() to create ID2D1Bitmaps from DXGI surfaces
  - Must use the same underlying Direct3D device

From Direct3D to Direct2D

```
ID2D1RenderTarget::CreateSharedBitmap(
    REFIID riid,
    void *data,
    D2D1_BITMAP_PROPERTIES *bitmapProperties,
    ID2D1Bitmap **bitmap );
```

- Must pass an IDXGISurface into data
- bitmapProperties has to match the format of the DXGI surface
  - Alpha is not included in this check

DIRECT2D/DIRECT3D RECOMMENDATIONS

- Device must support BGRA, D3D10.1
  - Falling back to Direct3D10Level9 can mitigate this
  - Special flag for device creation: D3D10_CREATE_DEVICE_BGRA_SUPPORT
- Render static content once and cache
- Minimize transitions between Direct3D and Direct2D by batching as much as possible

ADVANCED USE: MULTIPLE DEVICES

- What if your application needs to share resources across multiple devices?
- Answer: Synchronized Shared Surfaces

DXGI 1.1 SYNCHRONIZED SHARED SURFACES

- **IDXGIKEYEDMUTEX INTERFACE**
- **D3D10_RESOURCE_MISC_SHARED_KEYEDMUTEX**
- APIs supporting keyed mutex flag
  - `ID3D10Device1::CreateTexture1D/2D/3D`
  - `ID3D10Device1::CreateBuffer`

- MSDN Article: http://msdn.microsoft.com/en-us/library/ee913554(VS.85).aspx

DXGI 1.1 Synchronized Shared Surfaces cont'd…

```
//Specifying the keyed mutex flag
typedef enum D3D10_RESOURCE_MISC_FLAG {
    D3D10_RESOURCE_MISC_GENERATE_MIPS = 0x1L,
    D3D10_RESOURCE_MISC_SHARED = 0x2L,
    D3D10_RESOURCE_MISC_TEXTURECUBE = 0x4L,
    D3D10_RESOURCE_MISC_SHARED_KEYEDMUTEX = 0x10L,
    D3D10_RESOURCE_MISC_GDI_COMPATIBLE = 0x20L
} D3D10_RESOURCE_MISC_FLAG;
```

DXGI 1.1 Synchronized Shared Surfaces cont'd…

```
//Creating the synchronized shared surface with keyed
mutex flag
HRESULT ID3D10Device::CreateTexture2D(
  [in]   const D3D10_TEXTURE2D_DESC *pDesc,
  [in]   const D3D10_SUBRESOURCE_DATA *pInitialData,
  [out]  ID3D10Texture2D **ppTexture2D
);
```

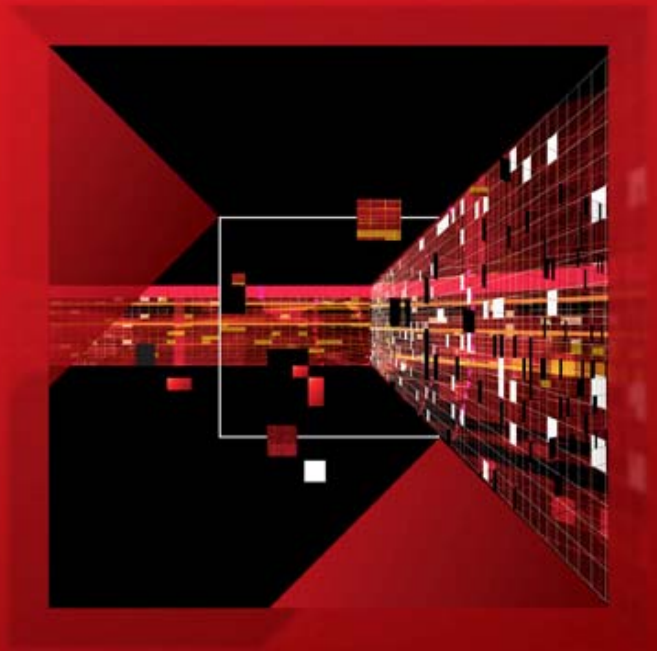
Obtain handle to Sync Shared Surface

```
//Use these methods for gaining access and relinquishing
device later

HRESULT IDXGIKeyedMutex::AcquireSync(
    [in] UINT64 Key,
    [in] DWORD dwMilliseconds
);

HRESULT IDXGIKeyedMutex::ReleaseSync(
    UINT64 Key
);
```

DEVELOPMENT RECOMMENDATIONS



ERROR HANDLING

- Dealing with device loss
  - Hybrid GPUs, USB Monitors and other “non standard” GPU configurations
  - Handling Terminal Server scenarios
  - Unplugging video hardware
  - Etc…
- Handle re-creation of resources on device loss
  - Handle the output of `HRESULT` to deal with runtime failures
  - `D2DERR_RECREATE_TARGET:` Discard all resources and re-create the target and resources

DIRECT2D DEBUG LAYER

- Intercepts calls from Direct2D
- Provides critical usage information as well as handy performance information
- Something failed? The debug layer will tell you.
- Leaves operation of Direct2D completely unchanged
  - May be a minor performance difference

LAYERS VS. CLIPS

- Layers (**ID2D1Layer**) let you manipulate a group of drawing operations.
  - Use a layer by "pushing" it onto a render target.
  - Subsequent operations by are directed to the layer.
  - “Pop" the layer to compose it to the render target.
- Only use layers when alpha-blending or non-rectangular content is needed.

LAYERS VS. CLIPS

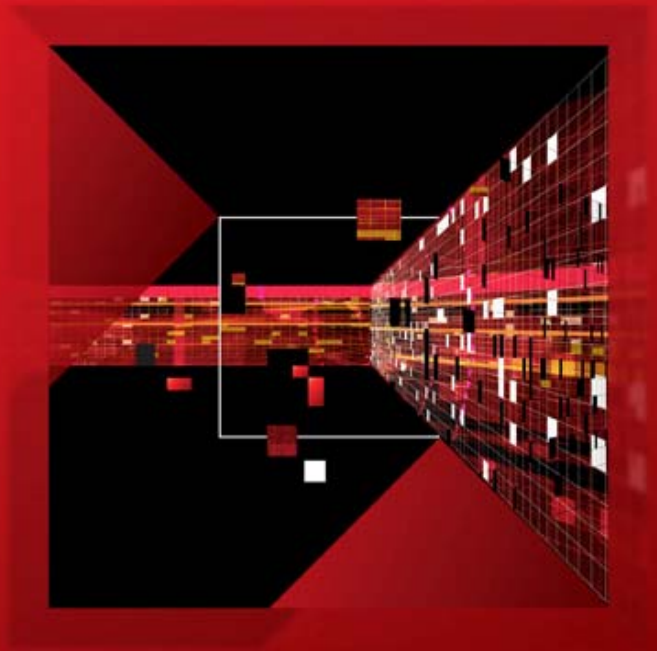
- Similar concept called “Clips”
  - Faster when you don’t need alpha blending or you have rectangular content
  - PushAxisAlignedClip/PopAxisAlignedClip
- Debug layer helps you know when to use one or the other:
  - D1111: Using Layer When Clip Is Sufficient

Turning on the Debug Layer

```
D2D1_FACTORY_OPTIONS options;
options.debugLevel =
    D2D1_DEBUG_LEVEL_INFORMATION;

hr = D2D1CreateFactory(
    D2D1_FACTORY_TYPE_SINGLE_THREADED,
    options,
    &m_pD2DFactory );
```

DIRECTWRITE

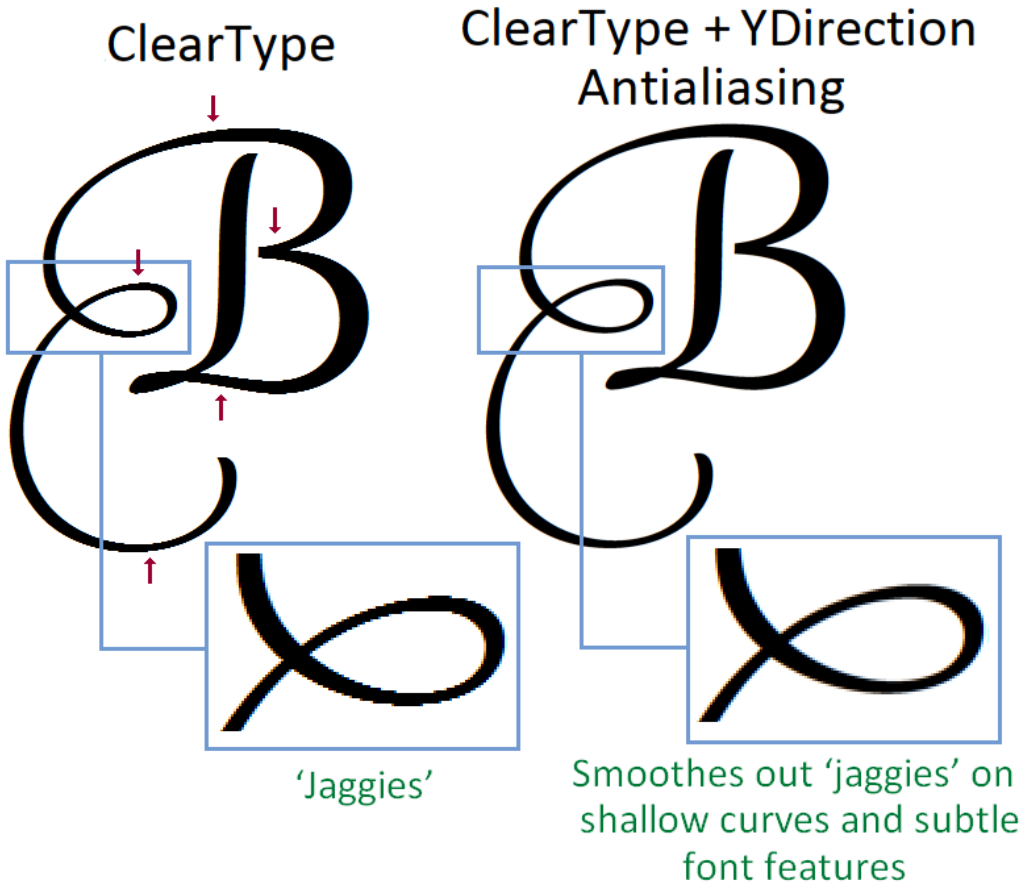


DIRECTWRITE

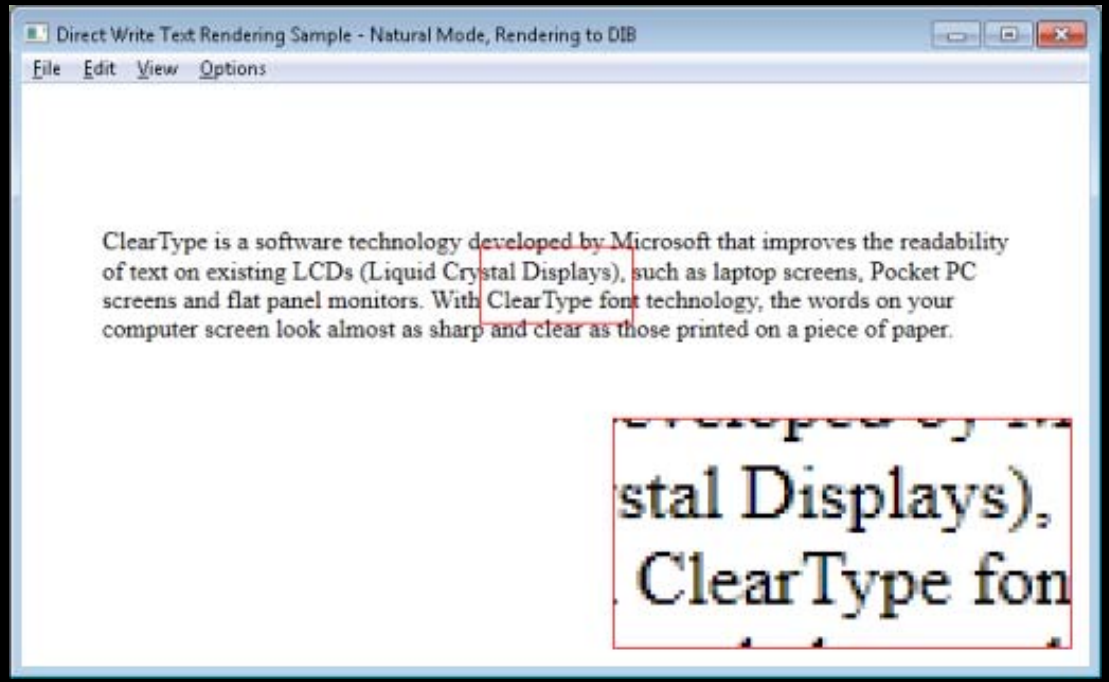
- DirectWrite – layered text layout & font processing system
  - –OpenType
  - –ClearType™
  - –E2E, easy to use rich Text Layout API
  - –Script Processing System
  - –Hardware accelerated text rendering using Direct2D

DIRECTWRITE RENDERING

PRECISE GLYPH SHAPES

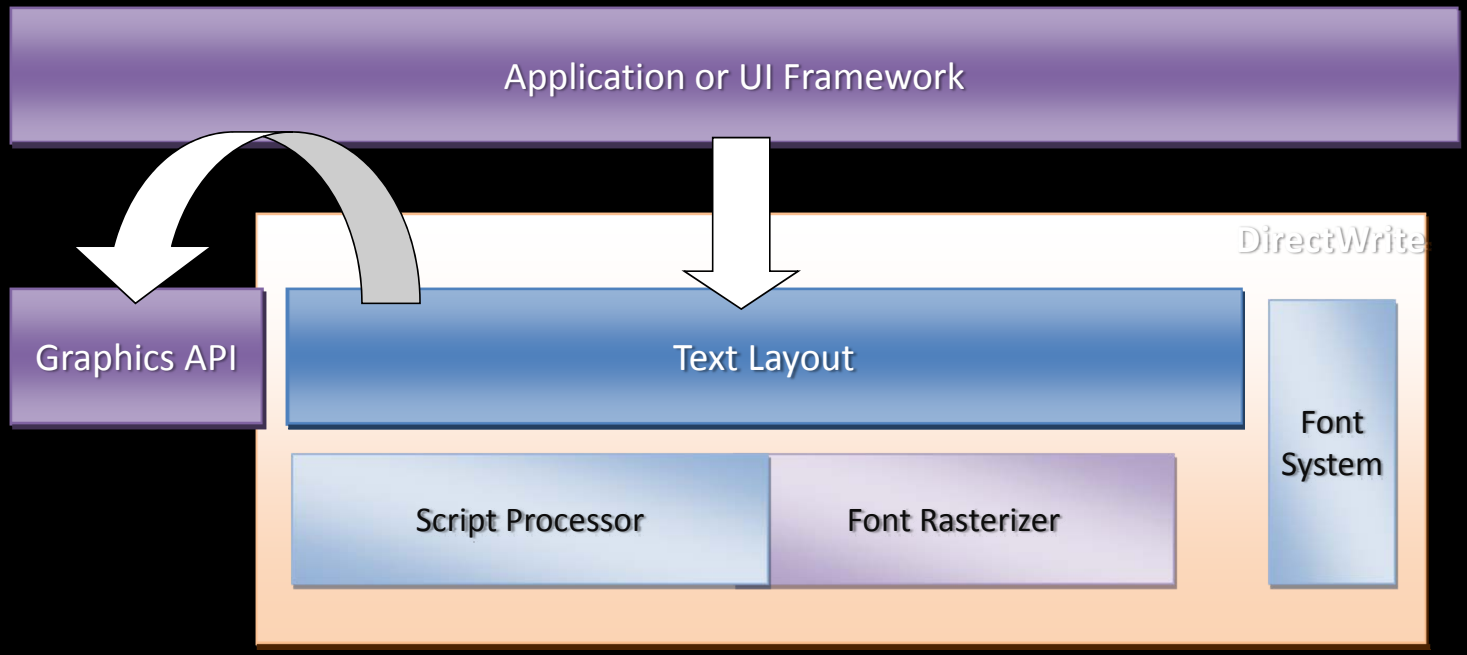


CLEAR TYPE


Direct Write Text Rendering Sample - Natural Mode, Rendering to DIB
File Edit View Options
ClearType is a software technology developed by Microsoft that improves the readability of text on existing LCDs (Liquid Crystal Displays), such as laptop screens, Pocket PC screens and flat panel monitors. With ClearType font technology, the words on your computer screen look almost as sharp and clear as those printed on a piece of paper.
stal Displays),
ClearType fon

DIRECTWRITE - MULTIPLE LAYERS OF FUNCTIONALITY

- Flexibility to adopt individual layers



USING DIFFERENT LAYERS OF DIRECTWRITE

- Using DirectWrite's Layout
  - Examples: Windows Live Applications
  - Developer uses IDWriteTextLayout and Direct2D's DrawTextLayout()
  - Minimal work, but less control
  - Great for UI scenarios (Control Labels, List boxes, etc.)
- Custom Layouts
  - Examples: IE9, Firefox
  - Developer uses DirectWrite's Text Analysis & Font System along with Direct2D's DrawGlyphRun() to layout and render text
  - Generally for Document Rendering scenarios

NEW IN WINDOWS 7 SP1

- Even Faster Text Rendering
- Improved Quality – light text on dark background
- Faster software fallback
- Also available on Windows Vista!

REFERENCES AND CONTACT INFORMATION

- References
  - Find more information about Direct2D, DirectWrite and all other DirectX APIs on MSDN: http://msdn.microsoft.com/en-us/directx/default.aspx
  - DirectX Developer Blog: http://blogs.msdn.com/b/directx/

- Get in touch!
  - dgtsig@microsoft.com

THANKS!

Disclaimer & Attribution

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.
The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. There is no obligation to update or otherwise correct or revise this information. However, we reserve the right to revise this information and to make changes from time to time to the content hereof without obligation to notify any person of such revisions or changes.

NO REPRESENTATIONS OR WARRANTIES ARE MADE WITH RESPECT TO THE CONTENTS HEREOF AND NO RESPONSIBILITY IS ASSUMED FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

ALL IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE ARE EXPRESSLY DISCLAIMED. IN NO EVENT WILL ANY LIABILITY TO ANY PERSON BE INCURRED FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
AMD, the AMD arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc. All other names used in this presentation are for informational purposes only and may be trademarks of their respective owners.

The contents of this presentation were provided by individual(s) and/or company listed on the title page. The information and opinions presented in this presentation may not represent AMD's positions, strategies or opinions. Unless explicitly stated, AMD is not responsible for the content herein and no endorsements are implied.